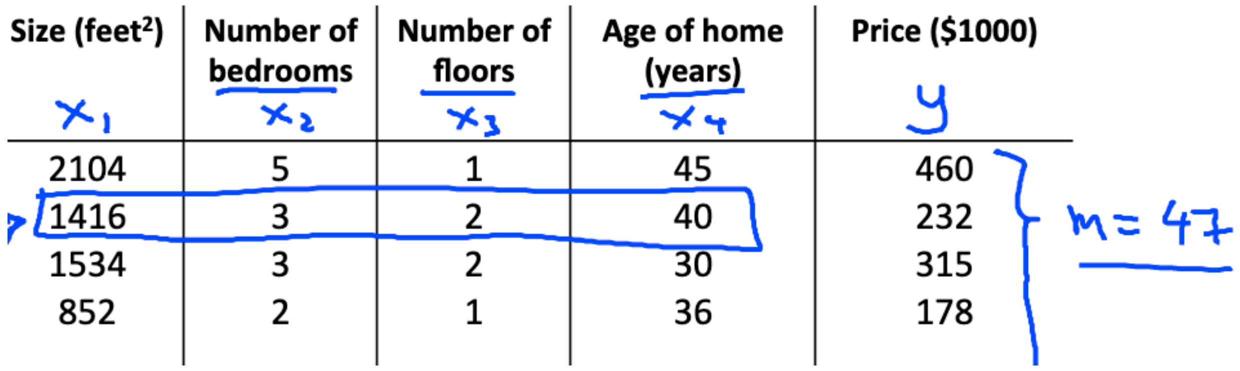
Multiple Features

Imagine you have a house dataset with many features or variables that we can use to predict the price. These features could be the number of bedrooms, the age of the home in years, and how far is it from the ocean. Thus, this dataset is giving us a lot of information to predict the price.

Notation

$x^{(i)}$ – the input features of the ith training example (will be a column vector)
$x_j{}^{(i)}$ – value of feature j in the ith training example.
m – the number of training examples.
n – the number of features.



Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_1	x_2	x_3	x_4	y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

$m = 47$

For example, the above dataset shows that $x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$ and $x_3{}^{(2)} = 2$. Because m = 47, the dataset has 47 rows.

Hypothesis Function

The multivariable hypothesis function is as follows:

$$h_0(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

We can use matrix vector multiplication to concisely represent the multivariable hypothesis function.

$$If\ we\ have\ x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \epsilon\ \mathbb{R}^{(n+1)}, \text{ and } \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \epsilon\ \mathbb{R}^{(n+1)},$$

$$then\ [\theta_0\ \theta_1 \cdots\ \theta_n] * \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Think of x_0 as $x_0 = 1$.

Gradient Descent for Multiple Variables

Given our hypothesis function, $h_0(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$, and the parameters that can be written as an $n + 1$ dimensional vector, θ, the cost function can be written as

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Thus, the Gradient descent is:
Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \frac{1}{m} * \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x_j^{(i)}$$
$$simultaneously\ update\ for\ j = 0, \ldots, j = n)$$

}

Feature Scaling

We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.

The way to prevent this is to modify the ranges of our input variables so that they are all roughly the same. Ideally:

$-1 \leq x_i \leq 1\ or\ -0.5 \leq x_i \leq 0.5$

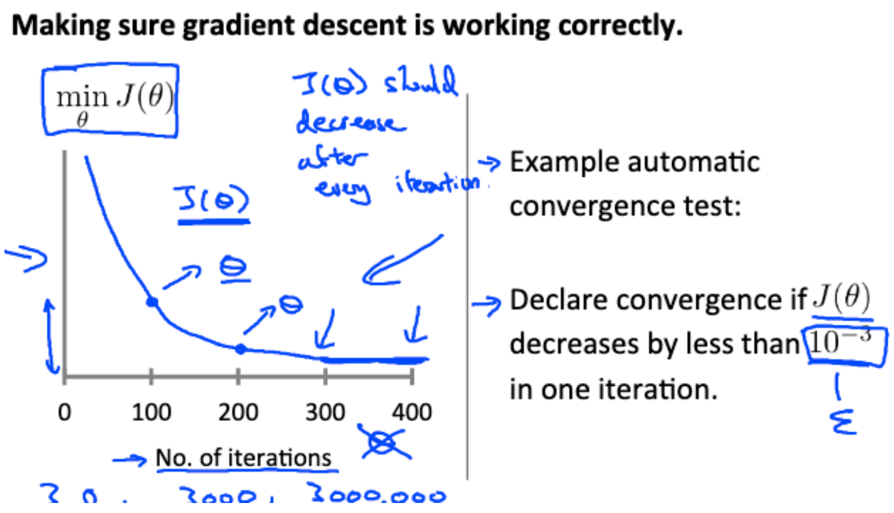
Two techniques to help with this are **feature scaling** and **mean normalization.** Feature scaling involves dividing the input values by the range of the input variable.

Mean normalization involves subtracting the average value for an input variable from the values for that input variable. To implement both these techniques, use the following formula:

$$x_i := \frac{x_i - \mu_i}{s_i}\ \ where\ s_i\ is\ the\ range\ of\ values\ or\ the\ standard\ deviation$$

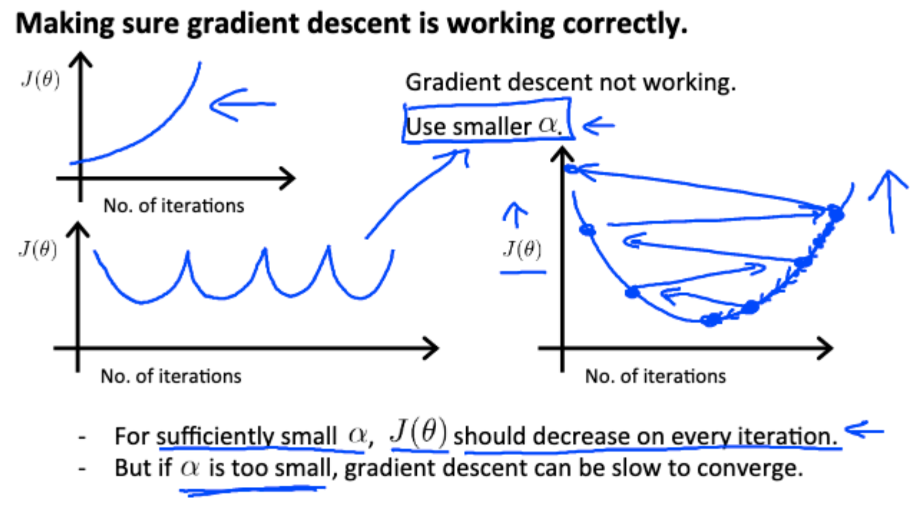
Practical tips for getting gradient descent to work and choosing the learning rate

It always helps to plot the number of iterations on the x-axis and observing how the cost function changes as the iterations increase. Basically, you run the algorithm for n iterations, and plot the value of the cost function after n iterations to get a plot that, hopefully, looks like the one below.



What this plot shows is that the value of the cost function after each iteration of gradient descent is decreasing. This means that the algorithm is working properly.

The plot also helps judge whether or not the gradient descent has converged. In this plot, the cost function is not decreasing much by the time the algorithm gets to 400 iterations.



The goal is to choose a sufficiently small learning rate. If too large, gradient descent will not converge.

Features and Polynomial Regression

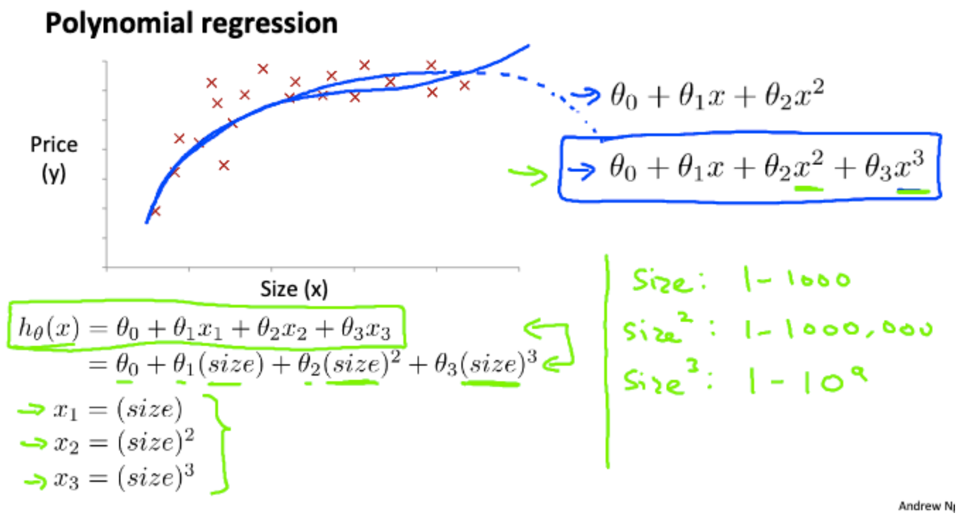
Suppose you have two features to predict home prices, $h_0(x) = \theta_0 + \theta_1 * frontage + \theta_2 * depth$

frontage
depth

In linear regression you don't necessarily have to use just the two features you had initially. You can create a new feature such house size which is product of frontage and depth.

Polynomial Regression
If a straight line does not fit the data well, we can change the behavior or curve of hypothesis function by making it a quadratic, cubic or a square root function.

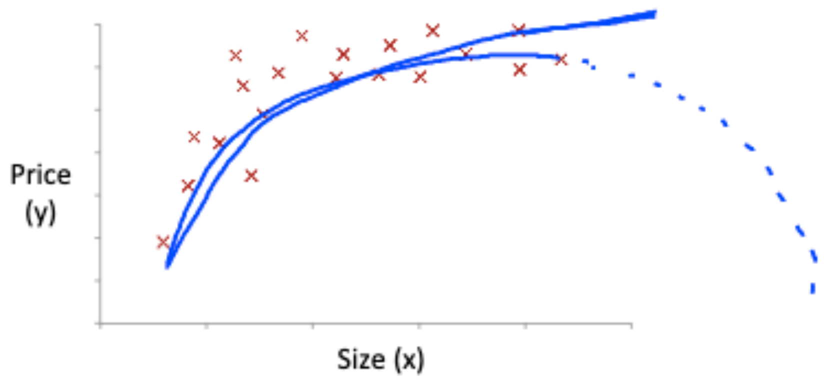
For example, we can fit a cubic model to learn a hypothesis function that will predict home prices based on home size.



Please know that feature scaling becomes very important when fitting quadratic or cubic functions to the data. For instance, if the size of the house ranges from one to one thousand square feet, then the size squared of the house will range from one to one million, and the third feature will range from one to 1,000,000,000.

Other feature choices
It looks like that you can get a better model by choosing to square root the size feature instead of a cubic model.

Choice of features



$$\rightarrow \; h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2(size)^2$$

$$\rightarrow \; \boxed{h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}}$$

Normal Equation
In linear regression, there is a second way of minimizing the cost function called the "Normal Equation" method. This method minimizes the cost function by explicitly taking the derivative with respect to the theta's and setting them to zero.

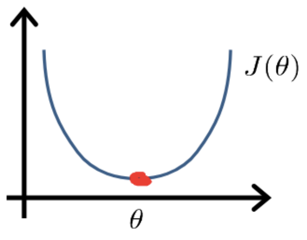
The figure below shows a very simple example.

Intuition: If 1D $(\underline{\theta \in \mathbb{R}})$

$$\rightarrow \; \underline{J(\theta) = a\theta^2 + b\theta + c}$$

$$\frac{d}{d\theta} J(\theta) = \dots \overset{set}{=} 0$$

Solve for θ



In the problem that we want to solve, $\theta \in \mathbb{R}^{(n+1)}$ and we want to minimize the cost function,

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

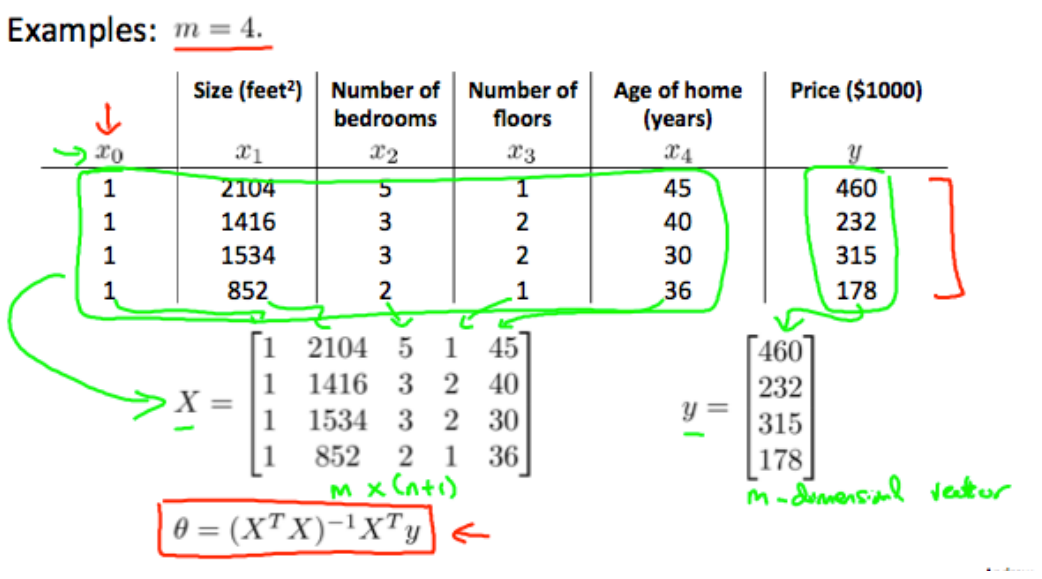
To explicitly solve for the optimal values of parameters theta,

$$\underline{\frac{\partial}{\partial\theta_j} J(\theta)} = \cdots \overset{set}{=} 0 \qquad \text{(for every } j\text{)}$$

Solve for $\underline{\theta_0, \theta_1, \dots, \theta_n}$

The normal equation method allows us to find the optimum theta without iteration. The formula for this equation is $\theta = (X^TX)^{-1}X^Ty$.

<u>**Example**</u>



Examples: $m = 4.$

	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$$\boxed{\theta = (X^TX)^{-1}X^Ty}$$

Initially, we had only 4 features, but we added an extra feature x_0 that always takes on a value of 1. In the picture, X is also called a design matrix. Note that the ith row of any design matrix is simply a 1 by (n+1) row vector. The values of this row vector are the values that the ith training example takes across all the features. Given that a dataset has m training examples, in general X is a m by (n+1) matrix.

There is **no need** to do feature scaling with the normal equation.

Advantages and Disadvantages of normal equation

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate inverse of X^TX
Works well when n is large	Slow if n is very large

With the normal equation, computing the inversion has complexity $\mathcal{O}(n^3)$. So if we have a very large number of features, the normal equation will be slow. In practice, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.

Also know that the normal equation method will not work for other sophisticated algorithms such as logistic regression or classification.

Normal Equation Non-invertibility

$(X^T X)$ will rarely be a singular matrix. A common reason for non-invertibility is having redundant features such as $size\ in\ feet^2$ and $size\ in\ m^2$. Redundant features that are closely related means that all the columns in your design matrix are not linearly independent.

A second reason is having too many features such that the number of features exceed the number of training examples. A solution to this problem is to delete one or more features.